Arduino – Programmierpraxis mit Feldvariablen

Arbeitsanregungen zu Grundlagen der Programmierung mit Feldvariablen

Aufgabe 0) Auf den Seiten 1 bis 3 der Informationsdatei **ST_I_10Arduino_Praxis_Feldvariable** sind Codefragmente abgebildet, in denen mit Hilfe von Schleifen Feldvariablen-Werte ein- und ausgegeben werden bzw. Berechnungen mit Feldvariablen durchgeführt werden.

a) Kombiniere diese Codefragmente, sodass lauffähige Arduino-Programme entstehen, speichere diese und teste sie (allenfalls mit Hilfe eines Arduino-Boards OHNE zusätzliche Schaltung).

b) Experimentiere sodann mit diesen Programmen, indem du das Befüllen der Feldvariablen veränderst, indem du durch Variation der Befehle `Serial.print` bzw. `Serial.println` das Layout der Ausgabe am seriellen Monitor veränderst und indem du die Berechnungen, die mit den in der Feldvariablen gespeicherten Werten durchgeführt werden, veränderst.

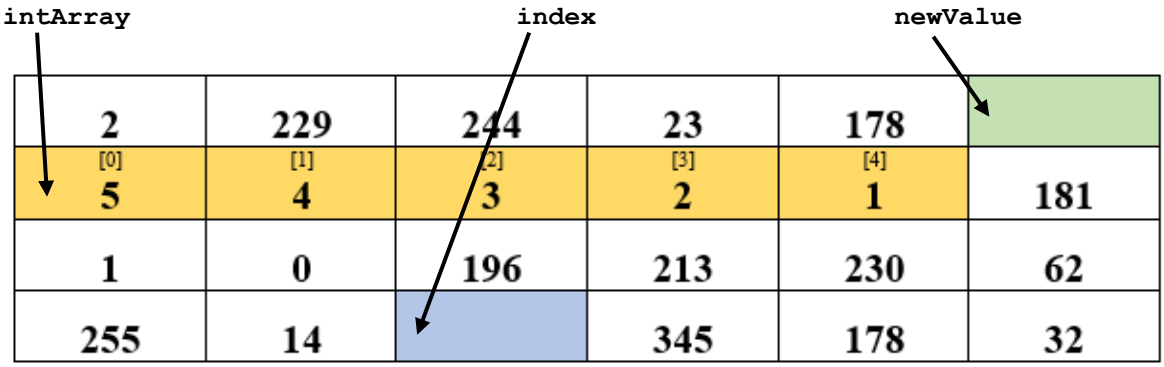
Aufgabe 1) In der Informationsdatei **ST_I_10Arduino__Praxis_Feldvariable** werden zwei Verfahren zur Analyse von Berechnungen mit Feldvariablen in Programmschleifen vorgestellt.

a) Analysiere Schritt für Schritt die Berechnungsschritte des zweiten Beispielcodes, der auf Seite 3 dieser Informationsdatei angegeben ist, mit Hilfe unseres „naiven Bildes“ vom Computerspeicher – du findest diesen Code nebenstehend nochmals abgebildet (beachte: **`intArray`** ist ein Array der Größe 5):

```
void loop() {
  int index;
  int newValue;

  index = 0;
  while(index < 4){
    newValue = intArray[index] + intArray[index + 1];
    Serial.print(" neuer Wert: ");
    Serial.println(newValue);
    index = index + 1;
  }
  Serial.println("");
  delay(5000);
}
```

Unmittelbar vor der Ausführung der Programmschleife soll die Situation im Speicher folgendermaßen aussehen (auch in den leergelassenen Speicherzellen steht irgendein „alter“ Wert):



Trage zunächst oben den fehlenden Wert der Variablen **`index`** ein (der Variablen **`newValue`** werden erst in der Schleife Werte zugewiesen!) und verwende die nachfolgenden „Speicherschablo-

nen“, um die Berechnungen mit den Werten der Feldvariablen Schritt für Schritt durchzuarbeiten.

Trage dazu jeweils die fehlende Werte ein und veranschauliche die Datenflüsse durch Pfeile:

Die Variable **index** hat zunächst den Wert _____,

daher wird die Schleife

```
while(index < 4){
  newValue = intArray[index] + intArray[index + 1];
  Serial.print(" neuer Wert: ");
  Serial.println(newValue);
  index = index + 1;
}
```

_____ (ausgeführt/nicht ausgeführt)

…zuerst wird gerechnet (`intArray[index] + intArray[index + 1];`):

Berechnung:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

… dann (`newValue = intArray[index] + intArray[index + 1];`) wird das errechnete Ergebnis in der Variablen **newValue** gespeichert und (`Serial.println(newValue);`) ausgegeben):

Ergebnis:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Ausgabe:

…und (`index = index + 1;`) der Wert der Variablen **index** verändert:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Die Variable **index** hat nun den Wert _____,

daher wird die Schleife

```
while(index < 4){
  newValue = intArray[index] + intArray[index + 1];
  Serial.print(" neuer Wert: ");
  Serial.println(newValue);
  index = index + 1;
}
```

_____ (ausgeführt/nicht ausgeführt)

…zuerst wird gerechnet (`intArray[index] + intArray[index + 1];`):

Berechnung:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

… dann (`newValue = intArray[index] + intArray[index + 1];`) wird das errechnete Ergebnis in der Variablen **newValue** gespeichert und (`Serial.println(newValue);`) ausgegeben):

Ergebnis:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Ausgabe:

…und (`index = index + 1;`) der Wert der Variablen **index** verändert:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Die Variable **index** hat nun den Wert _____,

daher wird die Schleife

```
while(index < 4){
  newValue = intArray[index] + intArray[index + 1];
  Serial.print(" neuer Wert: ");
  Serial.println(newValue);
  index = index + 1;
}
```

_____ (ausgeführt/nicht ausgeführt)

…zuerst wird gerechnet (`intArray[index] + intArray[index + 1];`):

Berechnung:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

… dann (`newValue = intArray[index] + intArray[index + 1];`) wird das errechnete Ergebnis in der Variablen **newValue** gespeichert und (`Serial.println(newValue);`) ausgegeben):

Ergebnis:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Ausgabe:

…und (`index = index + 1;`) der Wert der Variablen **index** verändert:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Die Variable **index** hat nun den Wert _____,

daher wird die Schleife

```
while(index < 4){
  newValue = intArray[index] + intArray[index + 1];
  Serial.print(" neuer Wert: ");
  Serial.println(newValue);
  index = index + 1;
}
```

_____ (ausgeführt/nicht ausgeführt)

…zuerst wird gerechnet (`intArray[index] + intArray[index + 1];`):

Berechnung:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

… dann (`newValue = intArray[index] + intArray[index + 1];`) wird das errechnete Ergebnis in der Variablen **newValue** gespeichert und (`Serial.println(newValue);`) ausgegeben):

Ergebnis:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Ausgabe:

…und (`index = index + 1;`) der Wert der Variablen **index** verändert:

2	229	244	23	178	
[0] 5	[1] 4	[2] 3	[3] 2	[4] 1	181
1	0	196	213	230	62
255	14		345	178	32

Die Variable **index** hat nun den Wert _____,

daher wird die Schleife

```
while(index < 4){
  newValue = intArray[index] + intArray[index + 1];
  Serial.print(" neuer Wert: ");
  Serial.println(newValue);
  index = index + 1;
}
```

_____ (ausgeführt/nicht ausgeführt)

b) Wie schon in der Informationsdatei **ST_10Ardunio_Praxis_Feldvariable** angedeutet, werden Programmschleifen (mit oder ohne Feldvariablen) schneller mit Hilfe einer Tabelle analysiert, wobei sämtliche Variablenwerte Schritt für Schritt und Zeile für Zeile notiert werden. Studiere zunächst das Beispiel in der Informationsdatei und verwende nachfolgend dieses „Tabellenverfahren“ zur nochmaligen Analyse des abgebildeten Codeteils:

```
void loop() {
  int index;
  int newValue;

  index = 0;
  while(index < 4){
    newValue = intArray[index] + intArray[index + 1];
    Serial.print(" neuer Wert: ");
    Serial.println(newValue);
    index = index + 1;
  }
  Serial.println("");
  delay(5000);
}
```

Durchlauf Nr.	index	intArray	newValue
0		{ 5, 4, 3, 2, 1 }	
1		{ 5, 4, 3, 2, 1 }	
2		{ 5, 4, 3, 2, 1 }	
3		{ 5, 4, 3, 2, 1 }	
4		{ 5, 4, 3, 2, 1 }	
5		{ 5, 4, 3, 2, 1 }	
6		{ 5, 4, 3, 2, 1 }	
…der Wert von **index** ist nicht mehr kleiner als ____, das Durchlaufen der Schleife wird beendet			

Aufgabe 2)

a) Eine Feldvariable wird gemäß nebenstehendem Code eines **setup**-Programmblocks mit Werten befüllt:

Mache dir klar, welche Werte am Ende des setup-Blocks in der Feldvariablen gespeichert sind – nutze dazu, wenn notwendig das „Tabellenverfahren“ zur Analyse von Programmschleifen.

Im loop-Programmblock werden mit den in der Feldvariablen gespeicherten Werten Berechnungen gemäß nebenstehend abgebildetem Code durchgeführt.
Finde (ohne das Programm am Arduino-Board ausführen zu lassen!) heraus, was hier berechnet wird. Verwende allenfalls die „naive Vorstellung“ vom Speicher oder das „Tabellenverfahren“.

```
int intArray[4];

void setup() {
  Serial.begin(9600); //für die Ausgabe
  int index;

  index = 0;
  while(index < 4){
    intArray[index] = 2 * (index + 1);
    index = index + 1;
  }
}

void loop() {
  int index;
  int ergebnis;

  ergebnis = 1;
  index = 0;
  while(index < 4){
    ergebnis = ergebnis * intArray[index];
    index = index + 1;
  }
  Serial.print(" Ergebnis: ");
  Serial.println(ergebnis);
  Serial.println("");
  delay(5000);
}
```

b) Mit der Feldvariablen, die durch den **setup**-Programmblock aus Aufgabenteil **a)** befüllt wurde, werden nun Berechnungen gemäß nebenstehendem **loop**-Programmblock durchgeführt:

```
void loop() {
  int index;
  int zwErg;
  float ergebnis;

  zwErg = 0;
  index = 0;
  while(index < 4){
    zwErg = zwErg + intArray[index];
    index = index + 1;
  }
  ergebnis = zwErg/(float)4;
  Serial.print(" Ergebnis: ");
  Serial.println(ergebnis);
  Serial.println("");
  delay(5000);
}
```

Dieser Programmcode hat zwei Neuerungen:

- Der Datentyp `float` wird verwendet, wenn in einer Variablen auch **andere als ganze Zahlen**, gespeichert werden sollen. Dabei werden `float`-Werte mit Dezimalpunkt (anstelle des bei uns gewohnten Dezimalkommas) geschrieben, also z.B. `3.125`.
- Wenn bei der Division ein `float`-Ergebnis erwartet wird, muss aufgepasst werden:

  In der Programmiersprache C ist das **Ergebnis einer Division** immer **ganzzahlig**, wenn sowohl **Dividend** als auch **Divisor** vom **Datentyp** `int` sind – der „Nachkommaanteil“ wird einfach „abgeschnitten“. Bei der Zuweisung `quotient = 21 / 5;` wird daher in der Variablen namens `quotient` nicht der Wert 4.2, sondern der Wert `4` gespeichert!
  Daher wird in obigem Programmcode die Division als `ergebnis = zwErg / (float) 4;` codiert:

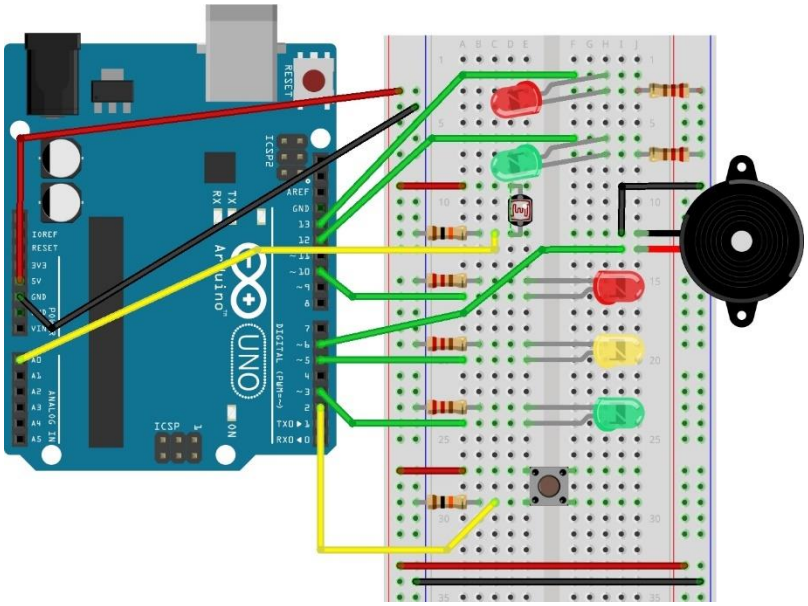
  Durch das vorangestellte `(float)` wird die Zahl 4 **für diese Rechnung** in eine Zahl vom Datentyp `float` umgewandelt und beim Ergebnis daher der „Nachkommaanteil“ nicht abgeschnitten.

  Alternativ dazu könnte auch `ergebnis = (float) zwErg / 4;` codiert werden, um die `int`-Variable `zwErg` **für diese Rechnung** in eine Variable vom Datentyp `float` umzuwandeln.

  Sobald zumindest einer der beiden Operanden – Dividend oder Divisor – vom Datentyp `float` ist, ist auch das Ergebnis der Division vom Datentyp `float`.

Mit diesen Informationen sollte es nun nicht zu schwer sein, herauszufinden, was im abgebildeten `loop`-Programmblock berechnet wird. Probier‘ es aus – du schaffst das sicher!

In der **folgenden Aufgabe** sollen die Programmcodes wieder auf einem Arduino-Board ausgeführt werden – dazu kannst du entweder wie in **Aufgabe 0** ein Arduino-Board ohne zusätzliche Schaltung benutzen oder die bereits mehrfach verwendete „Ampelschaltung“ (vgl. Abbildung) weiterverwenden.



Aufgabe 3)

a) Codiere nach dem Beispiel in der Informationsdatei **ST_I_10Arduino_Feldvariable** ein Programm,

- in dem im `setup`-Block eine `int`-Feldvariable der Größe 100 mit „schwer vorhersagbaren“ Zahlen befüllt wird;
- das einen selbst codierten Befehl-Programmblock zur Ausgabe der Werte einer `int`-Feldvariablen beliebiger Größe am seriellen Monitor beinhaltet;
- und das im `loop`-Block mit Hilfe dieses selbst codierten Befehlsblocks die Werte der `int`-Feldvariablen auch tatsächlich am seriellen Monitor ausgibt.

b) Erweitere das Programm aus Aufgabenteil **a)**, indem du einen selbst codierten Frage-Programmblock zur Berechnung der Summe aller in einem `int`-Feld gespeicherten Werte hinzufügst. Teste diesen Programmblock und gib die Summe der Feldwerte am seriellen Monitor aus.

c) Das arithmetische Mittel mehrerer Zahlen wird berechnet, indem die Summe dieser Zahlen durch die Anzahl der Zahlen dividiert wird (vgl. **Aufgabe 2 b)**). Erweitere das Programm aus Aufgabenteil **b)**, indem du einen selbst codierten Frage-Programmblock zur Berechnung des arithmetischen Mittels aller in einem `int`-Feld gespeicherten Werte hinzufügst – beachte, dass die Antwort dieses Frage-Programmblocks eine Zahl vom Datentyp `float` sein sollte! Teste diesen Programmblock und gib das arithmetische Mittel am seriellen Monitor aus.

d) Verändere den Code des Befehlsblocks aus Aufgabenteil **a)**, sodass über den Wert eines Parameters „gesteuert“ werden kann, ob die Werte der `int`-Feldvariablen vom kleinsten bist zum größten Index oder umgekehrt ausgegeben werden. Beispielsweise könnte die Überschriftenzeile dieses veränderten Befehlsblocks so aussehen:

```
void befehlPrintIntArray(int intArray[], int dim, int dir)
```

Dieser Befehl könnte dann (zum Beispiel) so funktionieren: Wenn der Wert des Parameters **`dir`** gleich 0 ist, sollen die Werte von Index 0 bis zum höchsten Index ausgegeben werden, ansonsten aber sollen die Werte vom höchsten Index bis zum Index 0 – also in umgekehrter Reihenfolge – ausgegeben werden.

…allfällige Erweiterungen, evtl. in einem weiteren Arbeitspaket (in pdf-Datei nicht enthalten):

Aufgabe 4) Morsen mit Eingabe über den Seriellen Monitor

Aufgabe 5) Lauflichter und Ausgabe mit binärer Codierung